Department of Mathematics
and Computer Science

Deep Learning

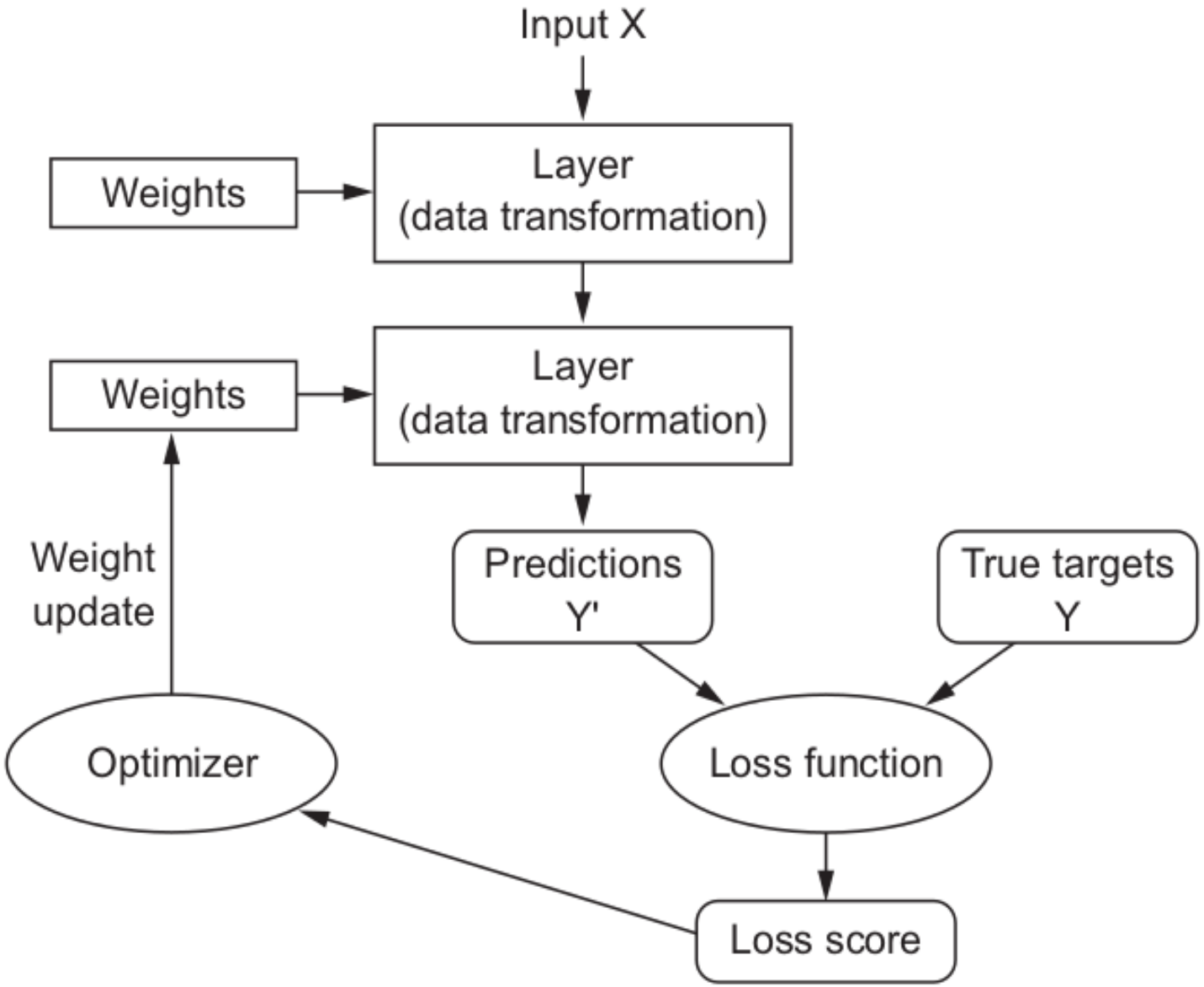
Introduction to PyTorch Part 2

Lukas Galke & Md Shamim Ahmed – Slides from Lucas Dyssel

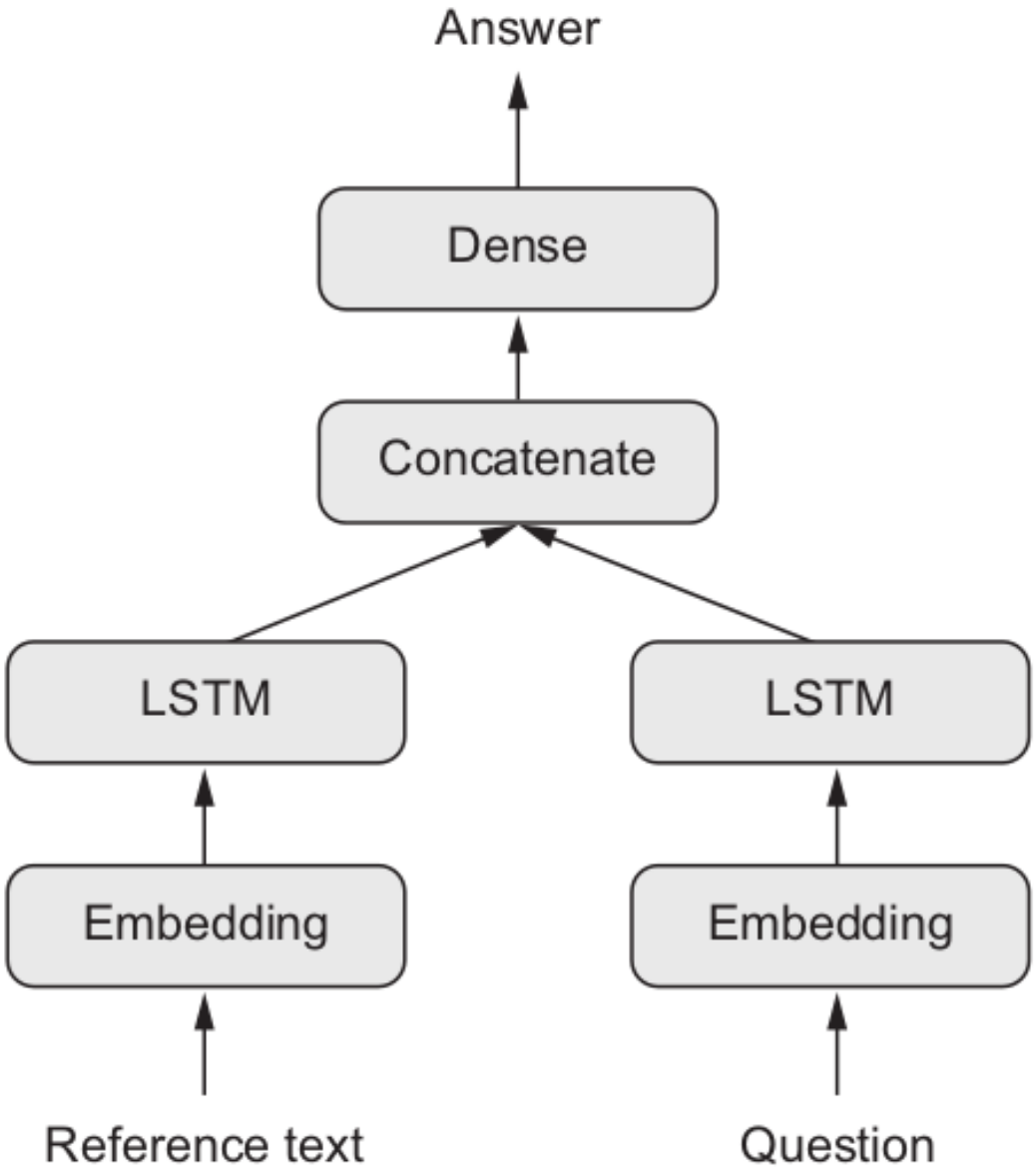
Fall 2023

SDU

Overview of Training a Neural Network



Multi Input Models

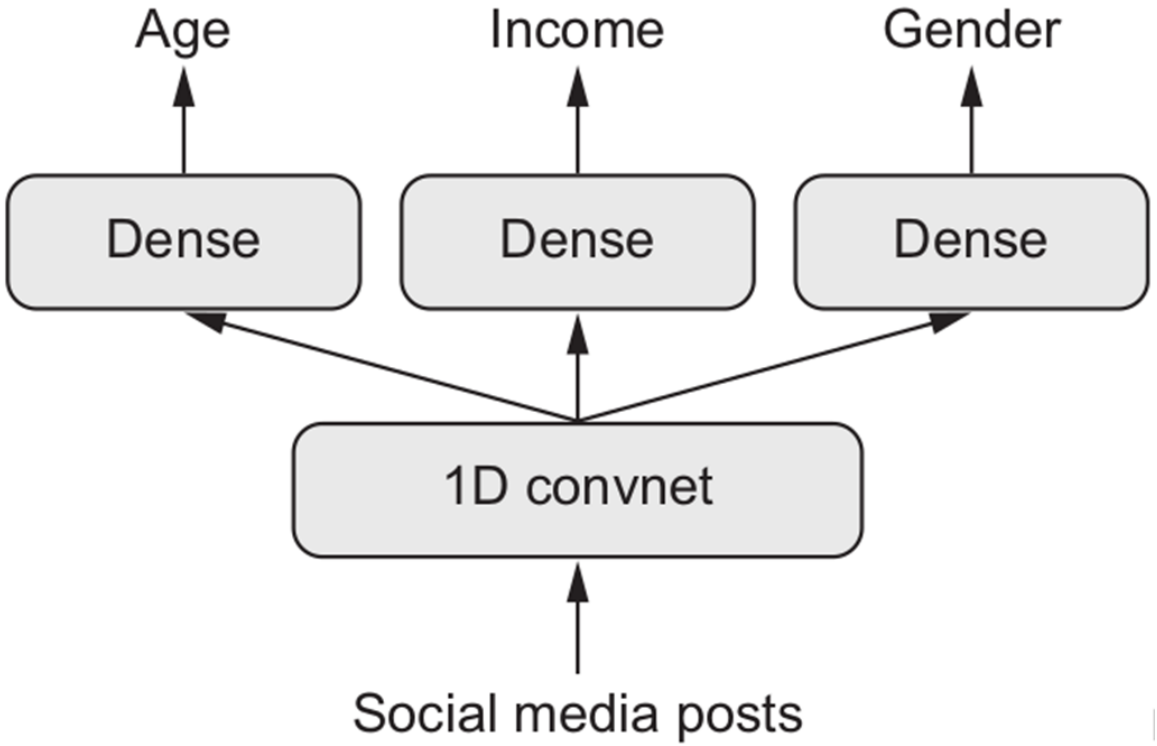


Multi Input Models

```
class NNetwork(nn.Module):
    def __init__(self):
        super(NNetwork, self).__init__()
        self.fc1 = nn.Linear(12, 16)
        self.fc2 = nn.Linear(16, 1)

    def forward(self, x1, x2):

        x = torch.cat((x1, x2), dim=1)

        x = torch.relu(self.fc1(x))
        x = torch.softmax(self.fc2(x))
        return x
```

- If your model takes two inputs, you simply concatenate the two input tensors along the second dimension
- Do however keep the dimensions in mind!
  - The input size of the first layer is no longer only dependent on 1 input, but 2
  - If the first input is of size 4 and the second input is of size 8, then the input size of the first fully connected layer should be $4 + 8 = 12$

Multi Output Models



Multi Output Models

```
class NNetwork(nn.Module):
    def __init__(self):
        super(NNetwork, self).__init__()
        self.fc1 = nn.Linear(10, 16)
        self.fc2 = nn.Linear(16, 1)
        self.fc3 = nn.Linear(16, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        output1 = torch.softmax(self.fc2(x))
        output2 = torch.sigmoid(self.fc3(x))
        return output1, output2
```

- If your model instead has two outputs, you simply add an extra “branch” with its own layers, activation function, etc.
- In this case **output1** uses **softmax** like before, but the new **output2** got its own fully connected layer and a sigmoid activation function

Early Stopping

```
best_loss = float('inf')
epochs_without_improvement = 0

############################
###### training loop ######
############################

# Early stopping
if loss < best_loss:
    best_loss = loss
    epochs_without_improvement = 0
else:
    epochs_without_improvement += 1

if epochs_without_improvement >= patience:
    print("Early stopping triggered.")
    exit()
```

- Before the training loop you want to initialize **best_loss** to very high value and **epochs_without_improvement** to 0
- After each epoch you want to check if your model improved by calculating the validation loss
  - This is the same as when we evaluate the model with respect to the test data, instead we just call “test” on the validation data each epoch rather than the end of the training
- If the loss is lower than the current **best_loss**, it means we should continue to train
- If it is not, we want to start counting
  - Once we reach some certain amount (in this case patience), we stop training and save the results
- This helps prevent overfitting and lowers wasted training time
- You can also use the package ignite to do early stopping

Using a Model for Predictions

```
import torchvision.models as models
from PIL import Image

model = models.resnet18(pretrained=True)
model.eval()

input_image = Image.open('path/to/your/
image.jpg')

with torch.no_grad():
    output = model(input_image)

predicted_idx = torch.argmax(output,
dim=1).item()
predicted_label = labels[predicted_idx]
```

- In this case we are loading the pretrained network ResNet18, you could also use your own trained network
- We then load an image using the PIL library
- We pass the image through ResNet18.
  - ResNet18 ends with a softmax, which means you get a probability distribution of what class the image belongs to
- We then use torch.argmax to get the index of the class which has the highest probability
- Lastly, we use this index to go through the list of classes the ResNet18 has trained on. This will allow us to find what class is in the image
  - Remember, a model can only predict on classes it has trained on. This is also the case for pretrained networks.

Deep Learning Workflow

- **Defining the problem and assembling a dataset**
  - What will your input data be?
  - What type of problem are you facing? Classification? Regression?
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- **Choosing a measure of success**
  - How do you measure if the model is successful
  - Not to be confused with the loss function which is something we use measure how close we are to the goal achieving our goal
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- **Decide on the evaluation protocol**
  - Hold-out validation
  - Cross-validation
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- **Prepare your data**
  - Clean data
  - Normalize data
  - Data augmentation
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- **Define a model**
  - Activation functions
  - Loss function
  - Optimization
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- **Scaling up: Make the model overfit**
  - Add layers
  - Make the layers bigger
  - Train for more epochs
- Regularizing your model and tuning you hyperparameters
- Finalize your final model

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- **Regularizing your model and tuning you hyperparameters**
  - This will take the most time
  - Add dropout
  - Try different architectures: add or remove layers
  - Add L1 and/or L2 regularization
- Finalize your final model

Regularization

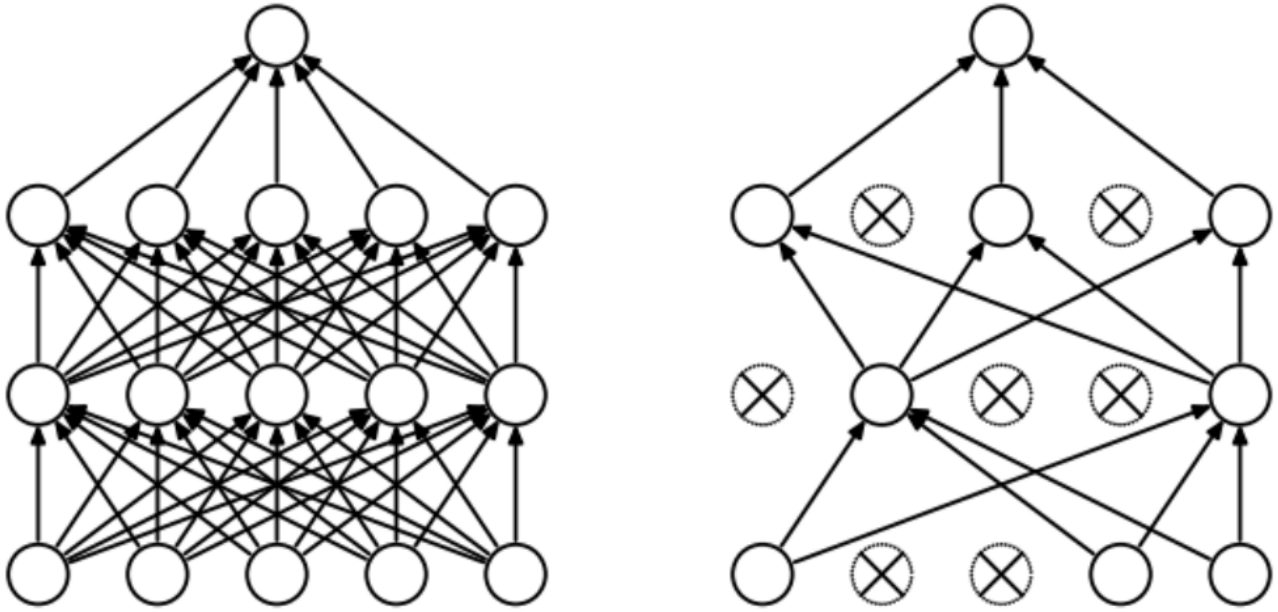
```
class MyNetwork(nn.Module):
    def __init__(self, input_size,
output_size):
        super(NNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 16)
        self.fc2 = nn.Linear(16, output_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.softmax(self.fc2(x))
        return x

net = MyNetwork(input_size=10, output_size=10)

######### INSIDE OF TRAINING LOOP #########
loss = criterion(outputs, targets)
loss += L2_lambda * torch.norm(net.fc1.weight,
p=2)
```

- **MyNetwork** is a standard neural network
- Adding the L2 regularization term to the loss function should happen inside of the training loop:
  - **L2_lambda** is a hyperparameter which determines the strength of of L2
  - **torch.norm(MyModel.fc1, p=2)** calculates the L2 norm (Euclidian norm) of the weights of the first fully connected layer
    - **p=1** is L1 regularization and **p=2** is L2 regularization
  - **loss += L2_lambda * torch.norm(MyModel.fc1, p=2)** the combined regularization term is added to the loss, penalizing large weight

Dropout

- **p (float)** – Probability of an element to be zeroed. Default: 0.5
- **inplace (bool)** – If set to True, will do this operation in-place. Default: False

```
torch.nn.Dropout(p=0.5, inplace=False)

# Example
class NNetwork(nn.Module):
    def __init__(self):
        super(NNetwork, self).__init__()
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        # Dropout is used inbetween two layers
        x = self.dropout(x)
```



(a) Standard Neural Net (b) After applying dropout.

Budhiraja, A. (2018, March 6). *Learning Less to Learn Better—Dropout in (Deep) Machine learning*. Medium. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

Deep Learning Workflow

- Defining the problem and assembling a dataset
- Choosing a measure of success
- Decide on the evaluation protocol
- Prepare your data
- Define a model better than base-line
- Scaling up: Make the model overfit
- Regularizing your model and tuning you hyperparameters
- **Finalize your final model**
  - Save and distribute the model